# A Graph-based Design Framework for Services

Antónia Lopes[1] and José Luiz Fiadeiro[2]

[1]Faculty of Sciences, University of Lisbon
Campo Grande, 1749–016 Lisboa, Portugal
mal@di.fc.ul.pt
[2]Department of Computer Science, Royal Holloway University of London
Egham TW20 0EX, UK
Jose.Fiadeiro@rhul.ac.uk

**Abstract.** Service-oriented systems rely on software applications that offer services through the orchestration of activities performed by external services procured on the fly when they are needed. This paper presents an overview of a graph-based framework developed around the notions of service and activity module for supporting the design of service-oriented systems in a way that is independent of execution languages and deployment platforms. The framework supports both behaviour and quality-of-service constraints for the discovery, ranking and selection of external services. Service instantiation and binding are captured as algebraic operations on configuration graphs.

## 1 Introduction

Service-oriented systems are developed to run on global computers and respond to business needs by interacting with services and resources that are globally available. The development of these systems relies on software applications that offer services through the orchestration of activities performed by other services procured on the fly, subject to a negotiation of service level agreements, in a dynamic market of service provision. The binding between the requester and the provider is established at run time at the instance level, i.e., each time the need for the service arises. Over the last few years, our research has addressed some challenges raised by this computing paradigm, namely:

(i) to understand the impact of service-oriented computing (SOC) on software engineering methodology;
(ii) to characterise the fundamental structures that support SOC independently of the specific languages or platforms that may be adopted to develop or deploy services;
(iii) the need for concepts and mechanisms that support the design of service-oriented applications from business requirements;
(iv) the need for mathematical models that offer a layer of abstraction at which we can capture the nature of the transformations that, in SOC, are operated on configurations of global computers;
(v) the need for an interface theory for service-oriented design.

As a result of (i) above, we identified two types of abstractions that are useful for designing service-oriented systems: business *activities* and *services*. Activities correspond to applications developed by business IT teams according to requirements provided by their organisation, e.g., the applications that, in a bank, implement the financial products that are made available to the customers. The implementation of activities may resort to direct invocation of components and can also rely on services that will be procured on the fly. Services differ from activities in that they are applications that are not developed to satisfy specific business requirements of an organisation; instead they are developed to be published in ways that they can be discovered by activities.

Taking into account this distinction, we developed a graph-based framework for the design of service-oriented systems at a level of abstraction that supports this "business-oriented" perspective. In this framework, services and activities are defined through *activity modules* and *service modules*, respectively. These modules differ in the type of interface and binding they provide to their clients, which in the case of activities is for direct invocation or static binding (e.g., human-computer interaction or system-to-system interconnections established at configuration time) and, in the case of services, for dynamic discovery and binding. Activity and service modules are graph-based primitives that define a workflow and the external services that may need to be procured and bound to in order to fulfil business goals. Behaviour and service-quality constraints can be imposed over the external services to be procured. These constraints are taken into account in the processes of discovery, ranking and selection.

The proposed design framework is equipped with a layered graph-based model for state configurations of global computers. Configurations are made to be *business reflective* through an explicit representation of the types of business activities that are active in the current state. This model captures the transformations that occur in the configuration of global computers when the discovery of a service is triggered, which results in the instantiation and binding of the selected service.

In this paper, we present an overview of this framework. In Sec. 2, we present the notions of service and activity modules, the cornerstone of our framework, grounded on an interface theory for service-oriented design. In Sec. 3, we present a model for state configurations of global computers that in Sec. 4 is used to provide the operational semantics of discovery, instantiation and binding. We conclude in Sec. 5 by pointing to other aspects of SOC that have been investigated within the framework.

## 2 Design Primitives for Service-oriented Systems

The design primitives we propose for service-oriented systems were inspired by the Service Component Architecture (SCA) [22]. As in SCA, we view SOC as providing an architectural layer that can be superposed over a component infrastructure – what is sometimes referred to as a service overlay. More concretely, we adopt the view that services are delivered by ensembles of components (or-

chestrations) that are able to bind dynamically to other services discovered at run time. For the purposes of this paper, the model that is used for defining orchestrations is not relevant. As discussed in Sec. 2.1, it is enough to know that we have a component algebra in the sense of [9] that makes explicit the structure of the component ensembles.

We illustrate our framework with a simplified credit service: after evaluating the risk of a credit request, the service either proposes a deal to the customer or denies the request; in the first case and if the proposal is accepted, the service takes out the credit and informs the customer of the expected transfer date. This activity relies on an external risk evaluator that is able to evaluate the risk of the transaction.

### 2.1 The Component Algebra

We see the ensembles of components that orchestrate services as networks in which *components* are connected through *wires*. For the purpose at hand, the nature of these components and the communication model is not relevant. The design framework is defined in terms of a set *COMP* of components, a set *PORT* of ports that components make available for communication with their environment, a set *WIRE* of wires for interconnecting pairs of ports, and a component algebra built around those elements. In the sequel we use $ports(c)$ and $ports(w)$ to denote, respectively, the set of ports of a component $c$ and the pair of ports interconnected by a wire $w$.

**Definition 1 (Component Net).** *A component net $\alpha$ is a tuple $\langle C, W, \gamma, \mu\rangle$ where:*

- *$\langle C, W\rangle$ is a simple finite graph: $C$ is a set of nodes and $W$ is a set of edges. Each edge is an unordered pair $\{c_1, c_2\}$ of nodes.*
- *$\gamma$ is a function assigning $\gamma_c \in COMP$ to every $c \in C$ and $\gamma_w \in WIRE$ to every $w \in W$.*
- *$\mu$ is a $W$-indexed family of bijections $\mu_w$ establishing a correspondence between $ports(\gamma_w)$ and the components $\{c_1, c_2\}$ interconnected by $w$, such that:*
  1. *For every $P \in ports(\gamma_w)$, $P \in ports(\mu_w(P))$.*
  2. *If $w' = \{c_1, c_3\}$ is an edge with $c_2 \neq c_3$, then $\mu_w(c_1) \neq \mu_{w'}(c_1)$.*

This definition reflects component-and-connector architectural configurations where the mapping $\mu$ defines the attachments between component ports and connector roles. Because in SOC communication is essentially peer-to-peer, we take all connectors to be binary. The fact that the graph is simple means that all interactions between two components are supported by a single wire and that no component can interact with itself. Through (2), ports of a component cannot be used in more than one connection.

The ports of a component net that are still available for establishing further interconnections, i.e., not connected to any other port, are called interaction-points:

**Definition 2 (Interaction-point).** *An interaction-point of a component net* $\alpha=\langle C, W, \gamma, \mu\rangle$ *is a pair* $\langle c, P\rangle$ *where* $c\in C$ *and* $P\in ports(\gamma_c)$ *such that there is no edge* $\{c, c'\}\in W$ *such that* $\mu_{\{c,c'\}}(c)=P$. *We denote by* $I_\alpha$ *the set of interaction-points of* $\alpha$.

Component nets can be composed through their interaction points via wires that interconnect the corresponding ports.

**Definition 3 (Composition of Component Nets).** *Let* $\alpha_1=\langle C_1, W_1, \gamma_1, \mu_1\rangle$ *and* $\alpha_2=\langle C_2, W_2, \gamma_2, \mu_2\rangle$ *be component nets such that* $C_1$ *and* $C_2$ *are disjoint,* $(w^i)_{i=1\ldots n}$ *a family of wires, and* $\langle c_1^i, P_1^i\rangle_{i=1\ldots n}$ *and* $\langle c_2^i, P_2^i\rangle_{i=1\ldots n}$ *families of interaction points of, respectively,* $\alpha_1$ *and* $\alpha_2$, *such that: (1) each* $w^i$ *is a wire connecting* $\{P_1^i, P_2^i\}$, *(2) if* $c_1^i = c_1^j$ *and* $c_2^i = c_2^j$ *then* $i = j$, *(3) if* $c_1^i = c_1^j$ *with* $i \neq j$, *then* $P_1^i \neq P_1^j$ *and (4) if* $c_2^i = c_2^j$ *with* $i \neq j$, *then* $P_2^i \neq P_2^j$. *The composition*

$$\alpha_1 \;\Big|\Big|^{i=1\ldots n}_{\langle c_1^i, P_1^i\rangle, w^i, \langle c_2^i, P_2^i\rangle}\; \alpha_2$$

*is the component net defined as follows:*

- *Its graph is* $\langle C_1 \cup C_2, W_1 \cup W_2 \cup \bigcup_{i=1\ldots n}\{c_1^i, c_2^i\}\rangle$.
- *Its functions* $\gamma$ *and* $\mu$ *coincide with that of* $\alpha_1$ *and* $\alpha_2$ *on the corresponding subgraphs. For the new edges,* $\gamma_{\{c_1^i, c_2^i\}}=w^i$ *and* $\mu_{\{c_1^i, c_2^i\}}(P_j^i) = c_j^i$.

In order to illustrate the notions just introduced, we take the algebra of *Asynchronous Relational Nets* (ARN) defined in [13]. In that algebra, components interact asynchronously through the exchange of messages transmitted through channels. Ports are sets of messages classified as incoming or outgoing. A component consists of a finite collection of mutually disjoint ports and a set of infinite sequences of sets of actions (traces), each action being the publication of an outgoing message or the reception of an incoming message (for simplicity, the data that messages may carry is ignored). Interconnection of components is established through channels – a set $P$ of messages and a set of traces. A wire consists of a channel and a pair of injections $\mu_i{:}P\rightarrow P_i$ that uniquely establishes connections between incoming and outgoing messages.

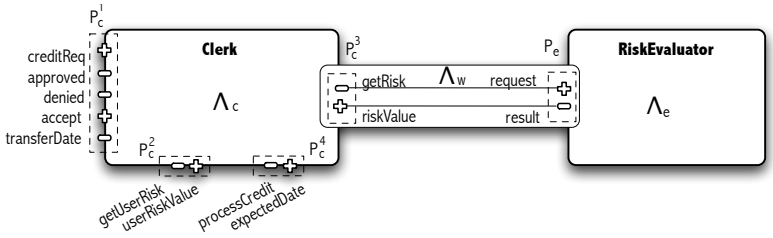


**Fig. 1.** An example of an ARN with two components connected through a wire.

Fig. 1 presents an example of an ARN with two components connected through a wire, which support part of the activities involved in the request of

a credit. The net has two nodes $\{c{:}Clerk, e{:}RiskEvaluator\}$ and a single edge $\{c, e\}{:}w_{ce}$.

The component *Clerk* has four ports. Its behaviour $\Lambda_c$ is as follows: after the delivery of the first *creditReq* message on port $P_c^1$, it publishes *getUserRisk* on port $P_c^2$ and waits for the delivery of *userRiskValue* in the same port; if the credit request comes from a known user, this may be enough for making a decision on the request and sending *approved* or *denied*; if not, it publishes *getRisk* on $P_c^3$ and waits for the delivery of *riskValue* for making the decision; after sending *approved* (if ever), *Clerk* waits for the delivery of *accept*, upon which it publishes *processCredit* on $P_c^4$ and waits for *expectedDate*; when this happens, it sends *transferDate*.

The component *RiskEvaluator* has a single port and its behaviour is quite simple: every time *request* is delivered, it publishes *result*. The wire $w_{ce}$ interconnects ports $P_c^3$ and $P_e$ and establishes that the publication of *getRisk* in $P_c^3$ will be delivered in $P_e$ under the name *request* and the publication of *request* in $P_e$ will be delivered in $P_c^3$ under the name *riskValue*.

The example presented in Fig. 1 can also be used to illustrate the composition of ARNs: this ARN is the composition of the two single-component ARNs defined by *Clerk* and *RiskEvaluator* via the wire $w_{ce}$.

### 2.2 The Interface Algebra

As discussed in the introduction, the interfaces of services and business activities need to specify the functionality that customers can expect as well as the dependencies that they may have on external services.

In our approach, a service interface identifies a port through which the service is provided (*provides-point*), a number of ports through which external services are required (*requires-points*) and a number of ports for those persistent components of the underlying configuration that the service will need to use once instantiated (*uses-points*). Activity interfaces are similar except that they have a *serves-point* instead of a provides-point. The differences are that the binding of provides and requires-points is performed by the runtime infrastructure whereas the binding of uses and serves-points has to be provided by developers.

In addition, interfaces describe the behavioural constraints imposed over the external services to be procured and quality-of-service constraints through which service-level agreements can be negotiated with these external services during matchmaking. The first are defined in terms of a logic while the latter are expressed through constraint systems defined in terms of c-semirings [5].

More concretely, we consider that sentences of a specification logic *SPEC* are used for specifying the properties offered or required. The particular choice of the specification logic – logic operators, their semantics and proof-theory – can be abstracted away. For the purpose of this paper, it is enough to know that the logic satisfies some structural properties, namely that we have available an entailment system (or $\pi$- institution) $\langle SIGN, gram, \vdash \rangle$ for *SPEC* [17, 11]. In this structure, *SIGN* is the category of signatures of the logic: signatures are sets of actions (e.g., the actions of sending and receiving a message $m$, which

we denote, respectively, by $m!$ and $m¡$) and signature morphisms are maps that preserve the structure of actions (e.g., their type, their parameters, etc). The grammar functor $gram{:}SIGN{\rightarrow}SET$ generates the language used for describing properties of the interactions in every signature. Notice that, given a signature morphism $\sigma{:}\Sigma{\rightarrow}\Sigma'$, $gram(\sigma)$ translates properties in the language of $\Sigma$ to the language of $\Sigma'$. Translations induced by isomorphims (i.e., bijections between sets of actions) are required to be conservative.

We also assume that $PORT$ is equipped with a notion of morphism that defines a category related to $SIGN$ through a functor $A{:}PORT{\rightarrow}SIGN$. The idea is that each port defines a signature that allows to express properties over what happens in that port. We use $A_P$ to denote the signature corresponding to port $P$. Moreover, we consider that every port $P$ has a *dual* port $P^{op}$ (e.g., the dual of a set of messages classified as incoming or outgoing is the same set of messages but with the dual classification).

For quality-of-service constraints, we adopt so-called soft constraints, which map each valuation of a set of variables into a space of degrees of satisfaction $A$. The particular soft-constraint formalism used for expressing constraints is not relevant for the approach that we propose. It is enough to know that we consider constraint systems defined in terms of a fixed c-semiring $S$, as defined in [5]:

- A *c-semiring* $S$ is a semiring of the form $\langle A, +, \times, 0, 1\rangle$ in which $A$ represents a space of degrees of satisfaction. The operations $\times$ and $+$ are used for composition and choice, respectively. Composition is commutative, choice is idempotent and 1 is an absorbing element (i.e., there is no better choice than 1). $S$ induces a partial order $\leq_S$ (of satisfaction) over $A$: $a \leq_S b$ iff $a + b = b$.
- A *constraint system* defined in terms of c-semiring $S$ is a pair $\langle D, V\rangle$ where $V$ is a totally ordered set (of variables), and $D$ is a finite set (domain of possible values taken by the variables).
- A *constraint* consists of a subset $con$ of $V$ and a mapping $def{:}D^{|con|}{\rightarrow}A$ assigning a degree of satisfaction to each assignment of values to the variables in $con$.
- The *projection* of a constraint $c$ over $I{\subseteq}V$, denoted by $c \Downarrow_I$, is $\langle def', con'\rangle$ with $con' = con \cap I$ and $def'(t') = \sum_{\{t \in D^{|con|} : t\downarrow^{con}_{con'} = t'\}} def(t)$, where $t\downarrow^Y_X$ denotes the projection of $Y$-tuple $t$ over $X$.

We start by defining a notion of service and activity interface.

**Definition 4 (Service and Activity Interface).** *An interface $i$ consists of:*

- *A set $I$ (of interface-points) partitioned into a set $I^{\rightarrow}$ with at most one element, which (if it exists) is called the* provides-point *and denoted by $i^{\rightarrow}$, a set $I^{\leftarrow}$ the member of which are called the* requires-points*, a set $I^{\uparrow}$ with at most one element, which (if it exists) is called the* serves-point *and denoted by $i^{\uparrow}$, a set $I^{\downarrow}$ the member of which are called the* uses-points*, such that either $I^{\rightarrow}$ or $I^{\uparrow}$ is empty.*
- *For every $r{\in}I$, a port $P_r$ and a consistent set of formulas $\Phi_r$ over $A_{P_r}$.*
- *For every $r{\in}I^{\leftarrow}{\cup}I^{\downarrow}$, a consistent set of formulas $\Psi_r$ over the amalgamated union of $A_{P_r}$ and $A_{P_r^{op}}$.*

– *A pair* $\mathcal{C} = \langle \mathcal{C}_{cs}, \mathcal{C}_{sla} \rangle$ *where* $\mathcal{C}_{cs}$ *is a constraint system* $\langle \mathcal{C}_D, \mathcal{C}_V \rangle$ *and* $\mathcal{C}_{sla}$ *is a set of constraints over* $\mathcal{C}_{cs}$.

*A* service interface *i is an interface such that* $I^{\uparrow}$ *is empty and* $I^{\rightarrow}$ *is a singleton. Conversely, an* activity interface *i is an interface such that* $I^{\rightarrow}$ *is empty and* $I^{\uparrow}$ *is a singleton.*

The formulas $\Phi_r$ at each interface-point $r$ specify the protocols that the element requires from external services (in the case of requires-points) or from other components (in the case of uses-points) and those that it offers to customer services (in the case of the provides-point) or to users (in the case of the serves-point). The formulas $\Psi_r$ express requirements on the wire through which the element expects to interact with $r$. $\mathcal{C}$ defines the constraints through which SLAs can be negotiated with external services during discovery and selection.

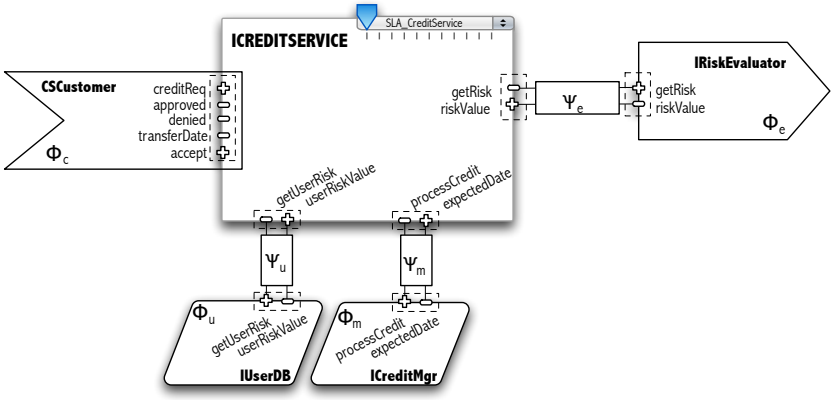


**Fig. 2.** An example of a service interface.

In Fig. 2, we present an example of an interface for a credit service using a graphical notation similar to that of SCA. On the left, we have a provides-point *CSCustomer* through which the service is provided; on the right, a requires-point *IRiskEvaluator* through which an external service is required; and, on the bottom, two uses-points through which the service connects to persistent components (a database that stores information about users and a manager of approved credit requests).

In this example, the specification of behavioural properties is defined in linear temporal logic. For instance, $\Phi_c$ includes $\Box(creditReq! \supset \Diamond(approved! \vee denied!))$ specifying that the service offers, in reaction to the delivery of the message *creditReq*, to reply by publishing either *approved* or *denied*. Moroever, $\Phi_c$ also specifies that if *accept* was received after the publication of *approved*, then *transferDate* will eventually be published. On the other hand, $\Phi_e$ only includes

$\Box(getRisk! \supset \Diamond riskValue!)$ specifying that the required service is asked to react to the delivery of *getRisk* by eventually publishing *riskValue*. $\Psi_e$ specifies that the wire used to connect the requires-point with the external service has to ensure that the transmission of both messages is reliable.

The quality-of-service constraints are defined in terms of the c-semiring $\langle [0,1], max, min, 0, 1\rangle$ of soft fuzzy constraints. *SLA_CreditService* declares three configuration variables – *c.amount* (the amount conceded to the customer), *e.fee* (the fee to be paid by the credit service to the risk evaluator service) and *e.cfd* (the confidence level of the risk evaluator) – and has two constraints: (1) the credit service targets only credit requests between 1000 and 10 000; (2) the fee $f$ to be paid to the risk evaluator must be less than 50, the confidence level $c$ must be greater than 0.9 and, if these conditions are met, the preference level is given by $\frac{c-0.9}{0.2} + \frac{50-f}{100}$.

Composition of interfaces is an essential ingredient of any interface algebra. As discussed before, activities and services differ in the form of composition they offer to their customers. In this paper, we focus on the notion of composition that is specific to SOC, which captures the binding of a service or activity with a required service.

**Definition 5 (Interface Match).** *Let $i$ be an interface, $r \in I^{\leftarrow}$ and $j$ a service interface. An* interface match *from $\langle i, r\rangle$ to $j$ consists of a port morphism $\delta : P^i_r \rightarrow P^j_{\rightarrow}$ such that $\Phi^j_{\rightarrow} \vdash \delta(\Phi^i_r)$ and a partial injective function $\rho : \mathcal{C}^i_V \rightarrow \mathcal{C}^j_V$. Interface $i$ is said to be* compatible *with service interface $j$ w.r.t. requires-point $r$ if (1) $I$ and $J$ are disjoint, (2) $blevel(\mathcal{C}^i_{sla} \oplus_\rho \mathcal{C}^j_{sla}) >_S 0$ and (3) there exists a match from $\langle i, r\rangle$ to $j$.*

An interface match defines a relation between the port of the requires-point $r$ of interface $i$ and the port of the provides-point of $j$ in such a way that the required properties are entailed by the provided ones. Moreover, the function $\rho$ identifies the configuration variables in the constraint systems of the two interfaces that are shared. The formulation of condition (2) above relies on a composition operator $\oplus_\rho$ that performs amalgamated unions of constraint systems and constraints, taking into account the shared configuration variables. These operations are defined as follows.

**Definition 6 (Amalgamation of Constraints).** *Let $\mathcal{S}_1 = \langle D_1, V_1\rangle$ and $\mathcal{S}_2 = \langle D_2, V_2\rangle$ be two constraint systems and $\rho : V_1 \rightarrow V_2$ a partial injective function.*

- *$\mathcal{S}_1 \oplus_\rho \mathcal{S}_2$ is $\langle D, V\rangle$ where $D$ is $D_1 \cup D_2$ and $V$ is $V_1 \oplus_\rho V_2$, the amalgamated union of $V_1$ and $V_2$. We use $\iota_i$ to denote the injection from $V_i$ into $V_1 \oplus_\rho V_2$.*
- *Let $c = \langle con, def\rangle$ be a constraint in $\mathcal{S}_i$. $\rho(c)$ is the constraint $\langle \iota_i(con), def'\rangle$ in $\mathcal{S}_1 \oplus_\rho \mathcal{S}_2$ where $def'(t)$ is $def(t)$ for $t \in D_i^{|con|}$ and 0 otherwise.*
- *Let $C_1$,$C_2$ be sets of constraints in, respectively, $\mathcal{S}_1$ and $\mathcal{S}_2$. $C_1 \oplus_\rho C_2$ is $\rho(C_1) \cup \rho(C_2)$.*

The consistency of a set of constraints $C$ in $\mathcal{S} = \langle D, V\rangle$ is defined in terms of the notion of best level of consistency as follows:

$$blevel(C) = \sum_{t \in D^{|V|}} \prod_{c \in C} def_c(t \downarrow^V_{con_c})$$

Intuitively, this notion gives us the degree of satisfaction that we can expect for $C$. We choose (through the sum) the best among all possible combinations (product) of all constraints in $C$ (for more details see [5]). $C$ is said to be consistent iff $blevel(C) >_S 0$. If a set of constraints $C$ is consistent, a valuation for the variables used in $C$ is said to be a *solution* for $C$ and can be also regarded as a constraint.

**Definition 7 (Composition of Interfaces).** *Given an interface $i$ compatible with a service interface $j$ w.r.t. $r$, a match $\mu{=}\langle\delta,\rho\rangle$ between $\langle i,r\rangle$ and $j$, and a solution $\Delta$ for $(\mathcal{C}^i_{cs} \oplus_\rho \mathcal{C}^j_{cs})\Downarrow_{\iota_j \circ \rho(\mathcal{C}^i_V)}$, the composition $i \parallel_{r:\mu,\Delta} j$ is $\langle K^{\rightarrow}, K^{\leftarrow}, K^{\uparrow}, K^{\downarrow}, P, \Phi, \Psi, \mathcal{C}\rangle$ where:*

- $K^{\rightarrow} = I^{\rightarrow}$, $K^{\leftarrow} = J^{\leftarrow} \cup (I^{\leftarrow} \setminus \{r\})$, $K^{\uparrow} = I^{\uparrow}$ *and* $K^{\downarrow} = I^{\downarrow} \cup J^{\downarrow}$.
- $P, \Phi, \Psi$ *coincides with* $P^i, \Phi^i, \Psi^i$ *and* $P^j, \Phi^j, \Psi^j$ *on the corresponding points.*
- $\mathcal{C} = \langle \mathcal{C}^i_{cs} \oplus_\rho \mathcal{C}^j_{cs}, (\mathcal{C}^i_{sla} \oplus_\rho \mathcal{C}^j_{sla}) \cup \{\Delta\}\rangle$.

Notice that the composition of interfaces is not commutative: the interface on the left plays the role of client and the one on the right plays the role of supplier of services.

### 2.3 Service and Activity Modules

A component net orchestrates a service interface by assigning interaction-points to interface-points in such a way that the behaviour of the component net validates the specifications of the provides-points on the assumption that it is interconnected to component nets that validate the specifications of the requires- and uses-points through wires that validate the corresponding specifications.

In order to reason about the behaviour of component nets we take the behaviour of components $c{\in}COMP$ and wires $w{\in}WIRE$ to be captured, respectively, by specifications $\langle A_c, \Phi_c\rangle$ and $\langle A_w, \Phi_w\rangle$ in $SPEC$ defining the language of components and wires to be the amalgamated union of the languages associated with their ports. Given a pair of port morphisms $\theta_1{:}P_1{\rightarrow}P'_1$ and $\theta_2{:}P_2{\rightarrow}P'_2$, we denote by $\langle\theta_1,\theta_2\rangle$ the unique mapping from the amalgamated sum of $A_{P_1}$ and $A_{P_2}$ to the amalgamated sum of $A_{P'_1}$ and $A_{P'_2}$ that commutes with $\theta_1$ and $\theta_2$.

Notice that nodes and edges denote *instances* of components and wires, respectively. Different nodes (resp. edges) can be labelled with the same component (resp. wire). Therefore, in order to reason about the properties of the component net as a whole we need to translate the properties of the components and wires involved to a language in which we can distinguish between the corresponding instances. We take the translation that uses the node as a prefix for the elements in their language. Given a set $A$ and a symbol $p$, we denote by $(p._-)$ the function that prefixes the elements of $A$ with '$p.$'. Note that prefixing defines a bijection between $A$ and its image $p.A$.

**Definition 8 (Component Net Properties).** *Let $\alpha = \langle C, W, \gamma, \mu\rangle$ be a component net. $A_\alpha = \bigcup_{c\in C} c.(A_{\gamma_c})$ is the language associated with $\alpha$ and $\Phi_\alpha$ is the union of, for every $c \in C$, the prefix-translation of $\Phi_{\gamma_c}$ by $(c._-)$ and, for every $w\in W$, the translation of $\Phi_{\gamma_w}$ by $\mu_w$, where, for $a\in A_P$, $\mu_w(a)=\mu_w(P).a$.*

The set $\Phi_\alpha$ consists on the translations of all the specifications of the components and wires using the nodes as prefixes for their language. Notice that because we are using bijections, these translations are conservative, i.e. neither components nor wires gain additional properties because of the translations. However, by taking the union of all such descriptions, new properties may emerge, i.e., $\Phi_\alpha$ is not necessarily a conservative extension of the individual descriptions.

**Definition 9 (Orchestration).** *An orchestration of an interface $i$ consists of:*

- *a component net $\alpha=\langle C, W, \gamma, \mu\rangle$ where $C$ and $I$ are disjoint;*
- *an injective function $\theta{:}I \to I_\alpha$ that assigns a different interaction-point to each interface-point; we write $r\xrightarrow{\theta}c$ to indicate that $\theta(r) = \langle c, P_c\rangle$ for some $P_c\in ports(\gamma_c)$;*
- *for every $r$ of $I^{\rightarrow} \cup I^{\uparrow}$, a port morphism $\theta_r{:}P_r\to P_{c_r}$ where $r\xrightarrow{\theta}c_r$;*
- *for every $r$ of $I^{\leftarrow} \cup I^{\downarrow}$, a port morphism $\theta_r{:}P_r^{op}\to P_{c_r}$ where $r\xrightarrow{\theta}c_r$;*

*If $i$ is a service interface, we require that*

$$\bigcup_{r\in I^{\leftarrow}\cup I^{\downarrow}} (\ r.\Phi_r \cup \mu_r(\Psi_r)\ ) \cup \Phi_\alpha \vdash c_{i^{\rightarrow}}.(\theta_{i^{\rightarrow}}(\Phi_{i_{\rightarrow}}))$$

*where $\mu_r(a) = r.a$ for $a\in A_{P_r}$ and $\mu_r(a) = c_r.\theta_r(a)$ for $a\in A_{P_r^{op}}$. We use $\alpha \lhd_\theta i$ to denote an orchestrated interface.*

Consider again the single-component ARN defined by *Clerk*. This ARN, together with the correspondences $CSCustomer \mapsto \langle Clerk, P_c^1\rangle$, $IRiskEvaluator \mapsto \langle Clerk, P_c^3\rangle$, $IUserDB \mapsto \langle Clerk, P_c^2\rangle$ and $ICreditMgr \mapsto \langle Clerk, P_c^4\rangle$, defines an orchestration for the service interface ICreditService. The port morphisms involved are identity functions. The traces in $\Lambda_c$ are such that they validate $\Phi_c$ on the assumption that *Clerk* is interconnected through *IRiskEvaluator*, *IUserDB* and *ICreditMgr* to component nets that validate, respectively, $\Phi_e$, $\Phi_u$ and $\Phi_m$ via wires that validate $\Psi_e$, $\Psi_u$ and $\Psi_m$.

Services are designed through service modules. These modules define an orchestrated service interface, the initialisation conditions for the components and the triggers for the requires-points (stating when external services need to be discovered). The proposed framework is independent of the language used for specifying initialisation conditions and triggers: we assume that we have available a set $STC$ of conditions and a set $TRG$ of triggers.

**Definition 10 (Service and Activity Module).** *A* service *(resp.* activity*)* module *consists of an orchestrated service (resp. activity) interface $\alpha \lhd_\theta i$ and*

*a pair of mappings $\langle trigger, init\rangle$ such that trigger assigns a condition in STC to each $r{\in}I^{\leftarrow}$ and init assigns a condition in STC to each c in the nodes of $\alpha$.*

*We use interface($M$), orch($M$) and $\theta^M$ to denote, respectively, $i$, $\alpha$ and $\theta$; $M^{\rightarrow}$ and $M^{\leftarrow}$ to denote, respectively, $i^{\rightarrow}$ and $I^{\leftarrow}$; $\mathcal{C}_{cs}(M)$ and $\mathcal{C}_{sla}(M)$ to denote, respectively, the constraint system and the set of constraints of $i$.*

The service interface ICREDITSERVICE orchestrated by *Clerk* together with a initialization condition for *Clerk* and a trigger condition for *IRiskEvaluator* define the service module CREDITSERVICE. We do not illustrate these conditions because their formulation depends on the formalism used for specifying the behaviour of components and wires (e.g., state machines, process calculi, Petri-nets). See [15, 16] for examples in SRML, a modelling language for SOC that we defined in the SENSORIA project.

**Definition 11 (Service Match).** *Let $M$ be a module and $r{\in}M^{\leftarrow}$. A service match for $M$ w.r.t. $r$ is a triple $\langle S, \mu, w\rangle$ where*

- *$S$ is a service module such that the set of nodes of orch($S$) is disjoint from that of orch($M$) and interface($M$) is compatible with interface($S$) w.r.t. $r$,*
- *$\mu$ is an interface match from $\langle$interface($M$), $r\rangle$ to interface($S$),*
- *$w$ is a wire connecting ports $\{P, P'\}$ such that $\Phi_w \vdash \langle\theta^M_r, \theta^S_{S^{\rightarrow}} \circ \rho\rangle(\Psi^M_r)$, assuming that $\theta^M(r){=}\langle c, P\rangle$ and $\theta^S(S^{\rightarrow}){=}\langle c', P'\rangle$.*

**Proposition and Definition 12 (Module Composition)** *Let $M$ be a service (resp. activity) module with interface $i$ and $r{\in}M^{\leftarrow}$; $\langle S, \mu, w\rangle$ a service match for $M$ w.r.t. $r$ with $\mu{=}\langle\delta, \rho\rangle$ and $j$ = interface($S$); $\Delta$ a solution for $(\mathcal{C}^i_{cs} \oplus_\rho \mathcal{C}^j_{cs})\Downarrow_{\iota_j \circ \rho(\mathcal{C}^i_V)}$. The composition $M \oplus_{r:\mu,w,\Delta} S$ is the service (resp. activity) module with:*

- *$(i \parallel_{r:\mu,\Delta} j) \rhd_\theta (orch(M) \;\big|\big|_{\theta^i(r),w,\theta^j(j^{\rightarrow})}\; orch(S))$, where $\theta$ coincides with $\theta^i$ on the interface-points inherited from $i$ and with $\theta^j$ on those inherited from $j$*
- *trigger and init have the conditions that are inherited from $M$ and $S$.*

*$M \oplus_{r:\mu,w} S$ is the composition in which no additional constraints are imposed on the external services, i.e., $M \oplus_{r:\mu,w,\emptyset} S$.*

Fig. 3 illustrates the elements involved in the composition of CREDITSERVICE (presented before) and RISKEVALSERVICE. The interface of this new service has the provides-point *RECustomer*, with $\Phi_r$ including $\Box(request_{¡} \supset \bigcirc result!)$, and its constraint system includes the configuration variables *r.fee* and *r.cfd* constrained by $r.fee{=}{-}3{+}\frac{3}{(1-r.cfd)}$. The orchestration of this service is provided by the single-component ARN with the component *RiskEvaluator* involved in the ARN presented before with $\kappa_{RECustomer}$ being the identity.

The match between the two services is given by the mappings $\delta$: *getRisk* $\mapsto$ *request*, *riskValue* $\mapsto$ *result* and $\rho$: *e.fee* $\mapsto$ *r.fee*, *e.cfd* $\mapsto$ *r.cfd*. The required property included in $\Phi_e$ translated by $\delta$ is $\Box(request_{¡} \supset \Diamond result!)$ which is trivially entailed by $\Phi_r$. For the composition of the two services, we take the wire $w_{ce}$

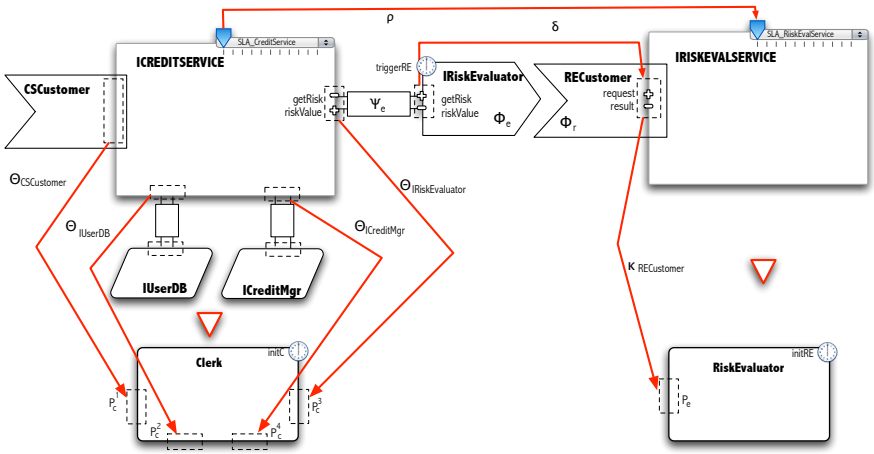


**Fig. 3.** Example of a service match.

also used in the ARN presented in Fig. 1 (its properties entail $\delta(\Psi_e)$) and the constraint $cfd = 0.9095$. This confidence level implies that the *fee* is approximately 30. This pair of values is one that provides the best level of consistency among the solutions for $(\mathcal{C}_{sla}(\textsc{CreditService}) \oplus_\rho \mathcal{C}_{sla}(\textsc{RiskEvalService})) \Downarrow_{\{fee,cfd\}}$.

The result of this composition is a service module whose interface has two uses- and two provides-points (inherited from CreditService), the variables *c.amount*, *fee* and *cfd* subject to the constraints inherited from the two interfaces and also $cfd = 0.9095 \wedge fee = 30.14917$. The service is orchestrated by the ARN presented in Fig. 1.

This example illustrates how the proposed notion of composition of services can be used for obtaining more complex services from simpler ones. The service provider of *IRiskEvaluator* was chosen at design-time as well as the SLA and the result of this choice was made available in a new service CreditService $\oplus$ RiskEvalService.

## 3 Business-reflective configurations

As mentioned before, component nets define configurations of global computers. In order to account for the way configurations evolve, it is necessary to consider the states of the configuration elements and the steps that they can execute. For this purpose, we take that every component $c \in COMP$ and wire $w \in WIRE$ of a component net may be in a number of states, the set of which is denoted by $STATE_c$ and $STATE_w$, respectively.

**Definition 13 (State Configuration).** *A* state configuration $\mathcal{F}$ *is a pair* $\langle \alpha, \mathcal{S} \rangle$, *where* $\alpha = \langle C, W, \gamma, \mu \rangle$ *is a component net and* $\mathcal{S}$ *is a configuration state, i.e., a mapping that assigns an element of* $STATE_c$ *to each* $c \in C$ *and of* $STATE_w$ *to each* $w \in W$.

A state configuration $\langle \alpha, \mathcal{S} \rangle$ may change in two different ways: (1) A state transition from $\mathcal{S}$ to $\mathcal{S}'$ can take place within $\alpha$ – we call such transitions *execution steps*. An execution step involves a local transition at the level of each component and wire, though some may be idle; (2) Both a state transition from $\mathcal{S}$ to $\mathcal{S}'$ and a change from $\alpha$ to another component net $\alpha'$ can take place – we call such transitions *reconfiguration steps*. In this paper, we are interested in the *reconfigurations steps* that happen when the execution of business activities triggers the discovery and binding to other services. In order to determine how state configurations of global computers evolve, we need a more sophisticated typing mechanism that goes beyond the typing of the individual components and wires: we need to capture the business activities that perform in a state configuration. We achieve this by typing the sub-configurations that, in a given state, execute the activities with activity module, thus making the configurations reflective.

Business configurations need also to include information about the services that are available in a given state (those that can be subject to procurement). We consider a space $\mathcal{U}$ of service and activity identifiers (e.g., URIs) to be given and, for each service and activity that is available, the configuration has information about its module and for each uses-point $u$: (i) the component $c_u$ in the configuration to which $u$ must be connected and (ii) a set of pairs of ports and wires available for establishing a connection with $c_u$. This information about uses-points of modules captures a 'direct binding' between the need of $u$ and a given provider $c_u$, reflecting the fact that composition at uses-points is integration-oriented. The multiplicity of pairs of ports and wires opens the possibility of having a provider $c_u$ serving different instances of a service at the same time.

We also consider a space $\mathcal{A}$ of business activities to be given, which can be seen to consist of reference numbers (or some other kind of identifier) such as the ones that organisations automatically assign when a service request arrives.

**Definition 14 (Business Configuration).** *A* business configuration *is* $\langle \mathcal{F}, \mathcal{P}, \mathcal{B}, \mathcal{C} \rangle$ *where*

- $\mathcal{F}$ *is a state configuration*
- $\mathcal{P}$ *is a partial mapping that assigns to services* $s \in \mathcal{U}$*, a pair*
$$\langle \mathcal{P}_M(s), \{\mathcal{P}^u(s) : u \in \mathcal{P}_M(s)^{\downarrow}\} \rangle$$
*where* $\mathcal{P}_M(s)$ *is a service module and* $\mathcal{P}^u(s)$ *consists of a node* $c^u$ *in* $\mathcal{F}$ *(i.e., a component instance) and a set of pairs* $\langle P_i^u, w_i^u \rangle$*, where each* $P_i^u$ *is a port of* $\gamma_{c^u}$ *distinct from the others and* $w_i^u$ *is a wire connecting* $P_i^u$ *and the port of* $u$ *that satisfies the properties* $\Psi^u$ *imposed by* $\mathcal{P}_M(s)$*. The services and activities in the domain of this mapping are those that are available in that state.*
- $\mathcal{B}$ *is a partial mapping that assigns an activity module* $\mathcal{B}(a)$ *to each activity* $a \in \mathcal{A}$ *(the workflow being executed by* $a$ *in* $\mathcal{F}$*). We say that the activities in the domain of this mapping are those that are active in that state.*
- $\mathcal{T}$ *is a mapping that assigns an homomorphism* $\mathcal{T}(a)$ *of graphs* $orch(\mathcal{B}(a)) \rightarrow \mathcal{F}$ *to every activity* $a \in \mathcal{A}$ *that is active in* $\mathcal{F}$*. We denote by* $\mathcal{F}(a)$ *the image of* $\mathcal{T}(a)$ *– the sub-configuration of* $\mathcal{F}$ *that corresponds to the activity* $a$*.*

Let us consider a configuration in which CREDITSERVICE (presented before) and CREDITACTIVITY are available. The latter is an activity that the same provider makes available in order to serve requests that are placed, not by other services, but by applications that interact with users (e.g., a web application that supports online credit requests). Suppose that the configuration also defines that the uses-points *IUserDB* and *ICreditMgr* of both interfaces should be connected, respectively, to *UserDB* and *CreditMgr* – a database of users and a manager of approved credit requests that are shared by all instances of this service and activity. The other elements of the business configuration are (partially) described in Fig. 4. It is not difficult to recognise that there are currently two active business activities – $A_{Alice}$ and $A_{Bob}$. Intuitively, both correspond to two instances of the same business logic (two customers requesting a credit using the same business activity) but at different stages of their workflow: one (launched by *BobUI*) is already connected to a risk evaluator (*BobREval*) while the other (launched by *AliceUI*) has still to discovery and bind to a risk evaluator service. The active computational ensemble of component instances that collectively pursue the business goal of each activity in the current state are highlighted through a dotted line.

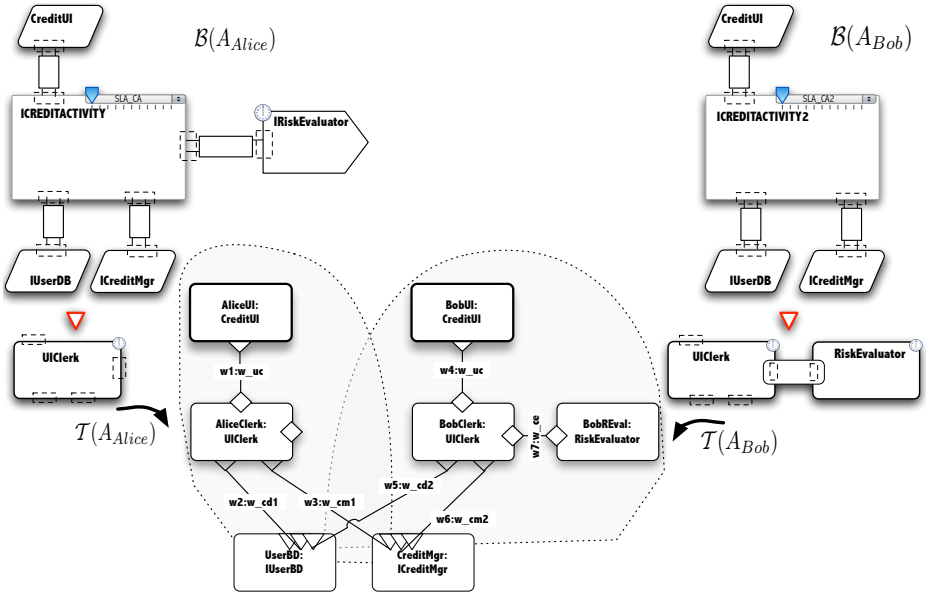


**Fig. 4.** Excerpt of a business configuration.

## 4 Service Discovery and Binding

Every activity module declares a triggering condition for each requires-point, which determines when a service needs to be discovered and bound to the current

configuration through that point. Let $\mathcal{L}=\langle \mathcal{F}, \mathcal{P}, \mathcal{B}, \mathcal{C} \rangle$ be the current business configuration. The discovery of a service for a given activity $a$ and requires-point $r$ of $\mathcal{B}(a)$ consists of several steps. First, it is necessary to find, among the services that are available in $\mathcal{L}$, those that are able to guarantee the properties associated with $r$ in $\mathcal{B}(a)$ and with which it is possible to reach a service-level agreement. Then, it is necessary to rank the services thus obtained, i.e., to calculate the most favourable service-level agreement that can be achieved with each $S$ – the contract that will be established between the two parties if $S$ is selected. The last step is the selection of one of the services that maximises the level of satisfaction offered by the corresponding contract.

**Definition 15 (discover$(M, r, \mathcal{P})$).** *Let $\mathcal{P}$ be a mapping as in Def. 14, $M$ an activity module and $r \in M^{\leftarrow}$.* **discover**$(M, r, \mathcal{P})$ *is the set of tuples $\langle s, \langle \delta, \rho \rangle, w, \Delta \rangle$ such that:*

1. *$s \in \mathcal{U}$ and $S=\mathcal{P}_M(s)$ is defined;*
2. *$\langle S, \langle \delta, \rho \rangle, w \rangle$ is a service match for $M$ w.r.t. $r$;*
3. *$\Delta$ is a solution for $(\mathcal{C}_{sla}(M) \oplus_\rho \mathcal{C}_{sla}(S)) \Downarrow_{\iota_S \circ \rho(\mathcal{C}_V^M)}$ and $blevel(\mathcal{C}_{sla}(M) \oplus_\rho \mathcal{C}_{sla}(S) \cup \{\Delta\})$ is greater than or equal to the value obtained for any other solution of that set of constraints;*
4. *$blevel(\mathcal{C}_{sla}(M) \oplus_\rho \mathcal{C}_{sla}(S) \cup \{\Delta\})$ is greater than or equal to the value obtained for any other tuple $\langle s', \delta', \rho', \Delta' \rangle$ satisfying the conditions 1-3, above.*

The discovery process for an activity module and one of its requires-points $r$ also provides us with a wire to connect $r$ with the provides-point of the discovered service. By Def. 11, this wire guarantees the properties associated with $r$ in $\mathcal{B}(a)$.

The process of binding an activity to a discovered service for one of its requires-points can now be defined:

**Definition 16 (Service Binding).** *Let $\mathcal{L}=\langle \mathcal{F}, \mathcal{P}, \mathcal{B}, \mathcal{T} \rangle$ be a business configuration with $\mathcal{F} = \langle \alpha, \mathcal{S} \rangle$, a an active business activity in $\mathcal{L}$ and $r \in \mathcal{B}(a)^{\leftarrow}$.*

- *If $\mathcal{F}(a) \models trigger_{\mathcal{B}(a)}(r)$ and* **discover**$(\mathcal{B}(a), r, \mathcal{L}) \neq \emptyset$*, then binding $\mathcal{B}(a)$ to $r$ using any of the elements in* **discover**$(\mathcal{B}(a), r, \mathcal{P})$ *is* enabled *in $\mathcal{L}$.*
- *Binding $\mathcal{B}(a)$ to $r$ using $\langle s, \langle \delta, \rho \rangle, w, \Delta \rangle \in$* **discover**$(\mathcal{B}(a), r, \mathcal{L})$*, and assuming that $\mathcal{P}_M(s)=S$, induces a business configuration $\langle \langle \alpha', \mathcal{S}' \rangle, \mathcal{P}, \mathcal{B}', \mathcal{T}' \rangle$ such that:*
  - *$\mathcal{B}'(x) = \mathcal{B}(x)$, if $x \neq a$ and $\mathcal{B}'(a)$ is the activity module $\mathcal{B}(a) \oplus_{r:\mu,w,\Delta} S$*
  - *if $\theta^{\mathcal{B}(a)}(r)=\langle c, P \rangle$ and $\theta^S(S^{\rightarrow})=\langle c', P' \rangle$, $\alpha'$ is $\alpha \parallel_\Xi \alpha_S$ where*
    - *$\alpha_S$ is a component net obtained by renaming the nodes in $orch(S)$ in such a way this set becomes disjoint from the set of nodes of $orch(\mathcal{B}(a))$,*
    - *$c''$ is the node in $\alpha_S$ corresponding to $c'$,*
    - *$\Xi=\{\langle \langle \mathcal{T}(c), P \rangle, w, \langle c'', P' \rangle \rangle, \langle \langle c^u, P_i^u \rangle, w_i^u, \theta^S(u) \rangle : u \in S^{\uparrow}\}$ where $c^u$ is the component identified by $\mathcal{P}^u(s)$, $P_i^u$ is a port in $\mathcal{P}^u(s)$ that is still available for connection and $w_i^u$ is the corresponding wire, as defined by $\mathcal{P}^u(s)$.*

- $\mathcal{S}'$ *coincides with* $\mathcal{S}$ *in the nodes of* $\alpha$ *and assigns, to every node* $c$ *in* $\alpha_S$*, a state that satisfies* $init_S(c)$*.*
- $\mathcal{T}'$ *is the homomorphism that results from updating* $\mathcal{T}$ *with the renaming from* $orch(S)$ *to* $\alpha_S$*.*

That is to say, new instances of components and wires of $S$ are added to the configuration while existing components are used for uses-interfaces, according to the direct bindings defined in the configuration.

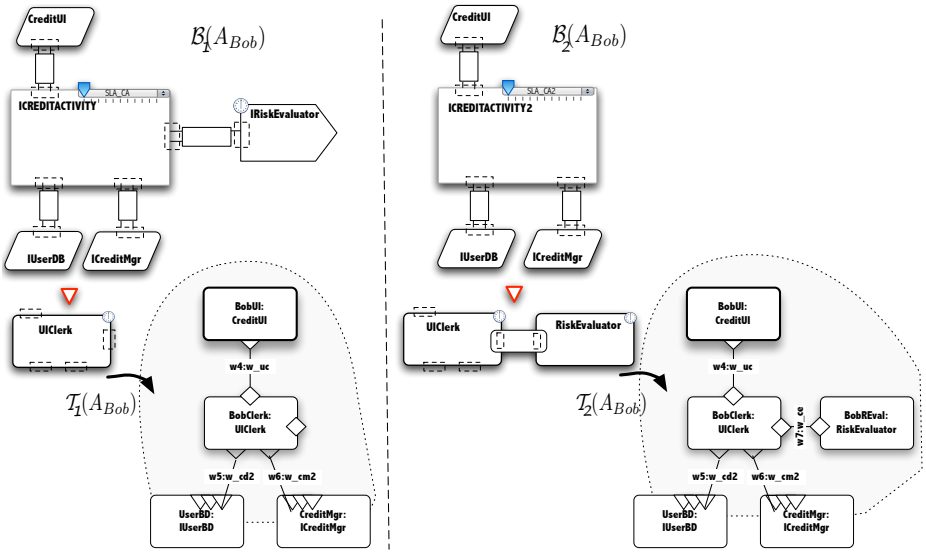


**Fig. 5.** Reconfiguration step induced by a service binding.

Figure 5 illustrates the binding process. On the left side is depicted part of the business configuration $\mathcal{L}_1$, before the activity $A_{Bob}$ has bound to a risk evaluator service (activity $A_{Alice}$ was not yet active at this time). On the right side is depicted the result of the binding of *IRiskEvaluator* to the service RiskEvalService using the service match presented in Fig. 3. This means that this service is, among all the services available in $\mathcal{P}_1$ that fit the purpose, one of the services that best fits the quality-of-service constraints. According to what discussed in Sec. 2.3, the contract established between the two parties is a confidence level of 0.9095 and a fee of 30.14917.

## 5 Conclusions

In this paper, we presented an overview of a graph-based framework for the design of service-oriented systems developed around the notion of service and activity module. Service modules, introduced in [14], were originally inspired by

concepts proposed in SCA [22]. They provide formal abstractions for composite services whose execution involves a number of external parties that derive from the logic of the business domain. Our approach has also been influenced by algebraic component frameworks for system modelling [10] and architectural modelling [2]. In those frameworks, components are, like services, self-contained modelling units with interfaces describing what they require from the environment and what they themselves provide. However, the underlying composition model is quite different as, unlike components, services are not assembled but, instead, dynamically discovered and bound through QoS-aware composition mechanisms. Another architectural framework inspired by SCA is presented in [25]. This framework is also language independent but its purpose is simply to offer a meta-model that covers service-oriented modelling aspects such as interfaces, wires, processes and data.

Our notion of service module also builds on the theory of interfaces for service-oriented design proposed in [13], itself inspired by the work reported in [9] for component based design. Henzinger and colleagues also proposed a notion of interface for web-services [4] but the underlying notion of composition, as in component-based approaches, is for integration.

We presented a mathematical model that accounts for the evolutionary process that SOC induces over software systems and used it to provide an operational semantics of discovery, instantiation and binding. This model relies on the mechanism of reflection, by which configurations are typed with models of business activities and service models. Reflection has been often used as a means of making systems adaptable through dynamic reconfiguration (e.g. [8, 19, 20]). A more detailed account of the algebraic properties of this model can be found in [15].

The presented framework was defined in terms of abstractions like *COMP*, *PORT* and *WIRE* so that the result was independent of the nature of components, ports and wires and of the underlying computation and communication model. In the same way, we have considered that the behavioural constraints imposed over the interface-points were defined in terms of a specification logic *SPEC*. A large number of formalisms have been proposed for describing each of these concepts in the context of SOC – e.g., process-calculi [7, 18, 26], automata-based models [3, 23] and models based on Petri-nets [21, 24]. An example of the instantiation of the framework is provided by the language SRML [12, 16], a modelling language of service-oriented systems we have developed in the context of SENSORIA project. This modelling language is equipped with a logic for specifying stateful, conversational interactions, and a language and semantic model for the orchestration of such interactions. Examples of quantitative and qualitative analysis techniques of service modules modelled in SRML can be found in [1, 6].

## References

1. J. Abreu, F. Mazzanti, J. L. Fiadeiro, and S. Gnesi. A model-checking approach for service component architectures. In *FMOODS/FORTE*, volume 5522 of *LNCS*,

pages 219–224, 2009.
2. R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1998.
3. B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing*, 8(1):46–54, 2004.
4. D. Beyer, A. Chakrabarti, and T. A. Henzinger. Web service interfaces. In A. Ellis and T. Hagino, editors, *WWW*, pages 148–159. ACM, 2005.
5. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
6. L. Bocchi, J. L. Fiadeiro, S. Gilmore, J. Abreu, M. Solanki, and V. Vankayala. A formal approach to modelling time properties of service oriented systems. In *Handbook of Research on Non-Functional Properties for Service-Oriented Systems: Future Directions*, Advances in Knowledge Management Book Series. IGI Global, in print.
7. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In R. De Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.
8. G. Coulson, G. S. Blair, P. Grace, F. Taïani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, 26(1), 2008.
9. L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In T. A. Henzinger and C. M. Kirsch, editors, *EMSOFT*, volume 2211 of *LNCS*, pages 148–165. Springer, 2001.
10. H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A component framework for system modeling based on high-level replacement systems. *Software and System Modeling*, 3(2):114–135, 2004.
11. J. L. Fiadeiro. *Categories for Software Engineering*. Springer, 2004.
12. J. L. Fiadeiro and A. Lopes. A model for dynamic reconfiguration in service-oriented architectures. In M. A. Babar and I. Gorton, editors, *ECSA*, volume 6285 of *LNCS*, pages 70–85. Springer, 2010.
13. J. L. Fiadeiro and A. Lopes. An interface theory for service-oriented design. In *FASE*, volume 6603 of *LNCS*, pages 18–33. Springer, 2011.
14. J. L. Fiadeiro, A. Lopes, and L. Bocchi. A formal approach to service component architecture. In *WS-FM*, volume 4184 of *LNCS*, pages 193–213, 2006.
15. J. L. Fiadeiro, A. Lopes, and L. Bocchi. An abstract model of service discovery and binding. *Formal Asp. Comput.*, 23(4):433–463, 2011.
16. J. L. Fiadeiro, A. Lopes, L. Bocchi, and J. Abreu. The SENSORIA reference modelling language. In M. Wirsing and M. M. Hölzl, editors, *Rigorous Software Engineering for Service-Oriented Systems*, volume 6582 of *LNCS*, pages 61–114. Springer, 2011.
17. J. L. Fiadeiro and A. Sernadas. Structuring theories on consequence. In *ADT*, volume 332 of *LNCS*, pages 44–72. Springer, 1987.
18. D. Kitchin, A. Quark, W. R. Cook, and J. Misra. The Orc programming language. In *FMOODS/FORTE*, volume 5522 of *LNCS*, pages 1–25, 2009.
19. F. Kon, F. M. Costa, G. S. Blair, and R. H. Campbell. The case for reflective middleware. *Commun. ACM*, 45(6):33–38, 2002.
20. M. Léger, T. Ledoux, and T. Coupaye. Reliable dynamic reconfigurations in a reflective component model. In *CBSE*, volume 6092 of *LNCS*, pages 74–92. Springer, 2010.

21. A. Martens. Analyzing web service based business processes. In *FASE*, volume 3442 of *LNCS*, pages 19–33. Springer, 2005.
22. OSOA. Service component architecture: Building systems using a service oriented architecture, 2005. White paper available from www.osoa.org.
23. J. Ponge, B. Benatallah, F. Casati, and F. Toumani. Analysis and applications of timed service protocols. *ACM Trans. Softw. Eng. Methodol.*, 19(4):11:1–11:38, Apr. 2010.
24. W. Reisig. Towards a theory of services. In R. Kaschek, C. Kop, C. Steinberger, and G. Fliedl, editors, *UNISCON*, volume 5 of *LNBIP*, pages 271–281. Springer, 2008.
25. W. van der Aalst, M. Beisiegel, K. van Hee, and D. Konig. An SOA-based architecture framework. *Journal of Business Process Integration and Management*, 2(2):91–101, 2007.
26. H. T. Vieira, L. Caires, and J. C. Seco. The conversation calculus: A model of service-oriented computation. In S. Drossopoulou, editor, *ESOP*, volume 4960 of *LNCS*, pages 269–283. Springer, 2008.